图文 104 深入探索BrokerOuter API是如何发送注册请求的?

详情 评论

深入探索BrokerOuter API是如何发送注册请求的?

继《从零开始带你成为JVM实战高手》后，阿里资深技术专家携新作再度出山，重磅推荐：

（点击下方蓝字试听）

《从零开始带你成为MySQL实战优化高手》

1、进入真正的注册请求方法去看看

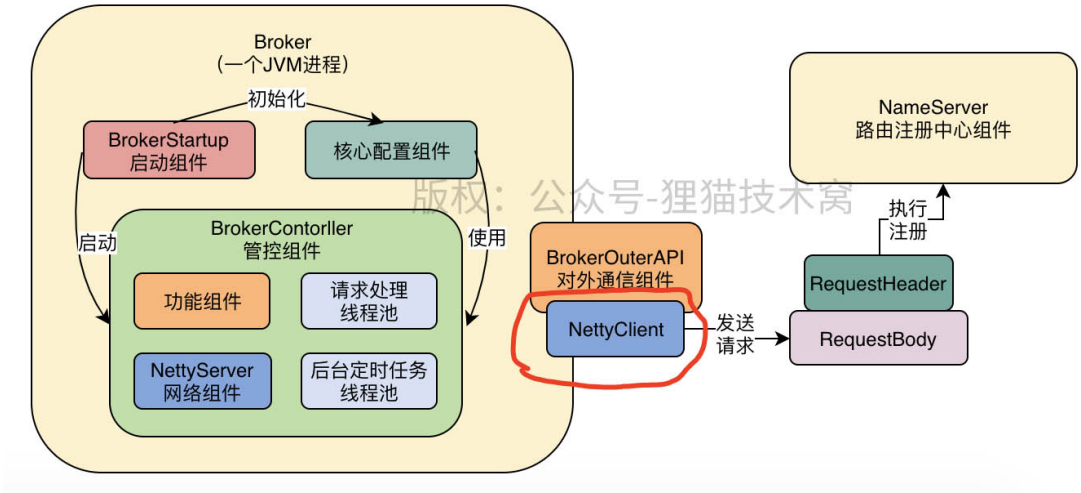
现在我们进入到真正的注册Broker的网络请求方法里去看看，其实入口就是下面这行代码：

```
RegisterBrokerResult result = registerBroker(
        namesrvAddr,oneway, timeoutMills,requestHeader,body);
```

进入这个方法之后，会看到下面的一段代码，我们可以看看，我写了详细的注释，大家仔细看看我写的注释。

```java
private RegisterBrokerResult registerBroker(final String namesrvAddr,
                                            final boolean oneway,
                                            final int timeoutMills,
                                            final RegisterBrokerRequestHeader requestHeader,
                                            final byte[] body
) throws Exception {

    // 两行代码大家看着有什么感触？是不是发现他搞了一个RemotingCommand
    // 然后他把请求头和请求体，都给弄到里面去了，这就封装了一个完整请求出来？
    RemotingCommand request =
            RemotingCommand.createRequestCommand(
                RequestCode.REGISTER_BROKER,
                requestHeader);

    request.setBody(body);

    // 这个oneway是说不用等待注册结果的意思，这里先不看了
    // 这属于特殊请求
    if (oneway) {

        try {
            this.remotingClient.
                invokeOneway(namesrvAddr, request, timeoutMills);
        } catch (RemotingTooMuchRequestException e) {
            // Ignore
        }
        return null;
    }

    // 真正的发送网络请求的逻辑是在这里
    // 大家注意一下这里面的RemotingClient是什么？不就是NettyClient吗！
    // 这个NettyClient就是用来发送网络请求出去的！
    RemotingCommand response = this.remotingClient.invokeSync(namesrvAddr, request, timeoutMills);

    // 下面就是在处理网络请求的返回结果了
    // 其实就是把注册请求的结果封装成了Result，保存了起来，并且返回了Result
    assert response != null;

    switch (response.getCode()) {

        case ResponseCode.SUCCESS: {

            RegisterBrokerResponseHeader responseHeader =
                (RegisterBrokerResponseHeader) response.decodeCommandCustomHeader(RegisterBrokerResponseHeader.class);

            RegisterBrokerResult result = new RegisterBrokerResult();
            result.setMasterAddr(responseHeader.getMasterAddr());
            result.setHaServerAddr(responseHeader.getHaServerAddr());

            if (response.getBody() != null) {
                result.setKvTable(KVTable.decode(response.getBody(), KVTable.class));
            }

            return result;
        }
        default:
            break;
    }
    throw new MQBrokerException(response.getCode(), response.getRemark());
}
```

看到这里，大家先看看下面的图，有没有发现最终的请求是基于NettyClient这个组件给发送出去的？大家看下面的红圈处。



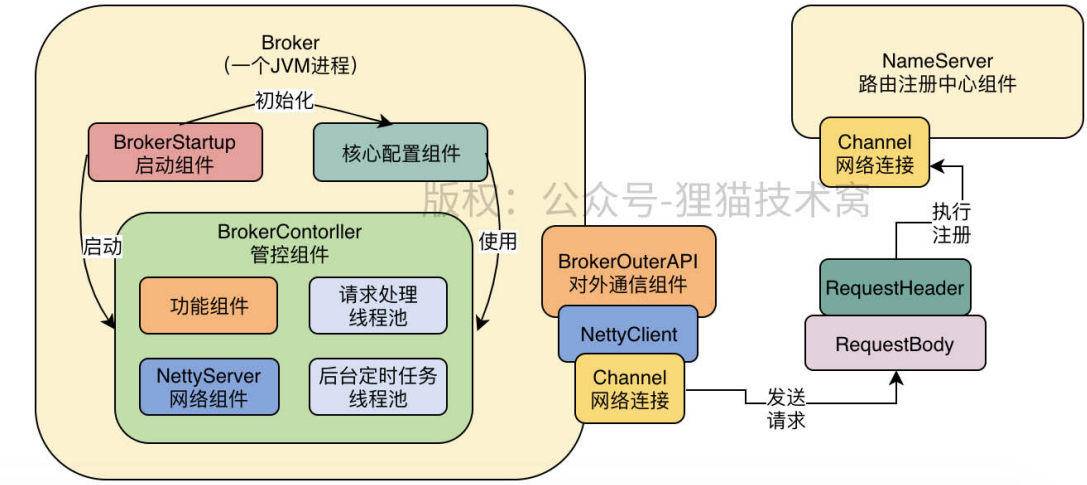
2、进入到NettyClient的网络请求方法中去看看

接着我们进入到NettyClient的网络请求方法中去看看，大家仔细看下面的代码，我都写了详细的注释了。

```
@Override
public RemotingCommand invokeSync(String addr,
                                  final RemotingCommand request,
                                  long timeoutMillis) {

    // 下面的代码就是获取一个当前时间
    long beginStartTime = System.currentTimeMillis();

    // 这行代码是很关键的！！！大家可以看到，在这里，他获取了一个Channel！
    // 大家知道这个Channel是什么吗？
    // 说白了，就是你这台Broker机器跟NameServer机器之间的一个连接
    // 所以你看下面传递进去了NameServer的地址，就是跟他建立了一个网络连接！
    // 而连接建立之后，就是用一个Channel来代表他的！
    final Channel channel = this.getAndCreateChannel(addr);

    // 下面的意思就是说，如果你跟NameServer之间的网络连接是ok的
    // 那么就可以发送请求了
    if (channel != null && channel.isActive()) {
        try {
            // 下面一大坨代码不用管，计算一些时间的
            doBeforeRpcHooks(addr, request);
            long costTime = System.currentTimeMillis() - beginStartTime;
            if (timeoutMillis < costTime) {
                throw new RemotingTimeoutException("invokeSync call timeout");
            }
            // 这里是真正发送网络请求出去的地方
            RemotingCommand response = this.invokeSyncImpl(channel, request, timeoutMillis - costTime);

```

```
            // 下面的也不用管，就是发送请求之后干点别的什么，最后返回结果了
            doAfterRpcHooks(RemotingHelper.parseChannelRemoteAddr(channel), request, response);
            return response;

        } catch (RemotingSendRequestException e) {

            log.warn("invokeSync: send request exception, so close the channel[{}]", addr);
            this.closeChannel(addr, channel);
            throw e;
        } catch (RemotingTimeoutException e) {
            if (nettyClientConfig.isClientCloseSocketIfTimeout()) {
                this.closeChannel(addr, channel);
                log.warn("invokeSync: close socket because of timeout, {}ms, {}", timeoutMillis, addr);
            }
            log.warn("invokeSync: wait response timeout exception, the channel[{}]", addr);
            throw e;
        }
    } else {
        this.closeChannel(addr, channel);
        throw new RemotingConnectException(addr);
    }
}
```

通过上面代码的分析，我现在在下面的图里，给大家再次加入一些东西，我通过Channel这个概念，表示出了Broker和NameServer之间的一个网络连接的概念，然后通过这个Channel就可以发送实际的网络请求出去！



3、如何跟NameServer建立网络连接?

接着我们进入上面的this.getAndCreateChannel(addr)这行代码看看，他是如何跟NameServer之间建立实际的网络连接的?

大家看下面的代码，下面的代码就是先从缓存里尝试获取连接，如果没有缓存的话，就创建一个连接。

```
private Channel getAndCreateChannel(final String addr) {

    if (null == addr) {
        return getAndCreateNameserverChannel();
    }
    ChannelWrapper cw = this.channelTables.get(addr);

    if (cw != null && cw.isOK()) {
        return cw.getChannel();
    }
    return this.createChannel(addr);
}
```

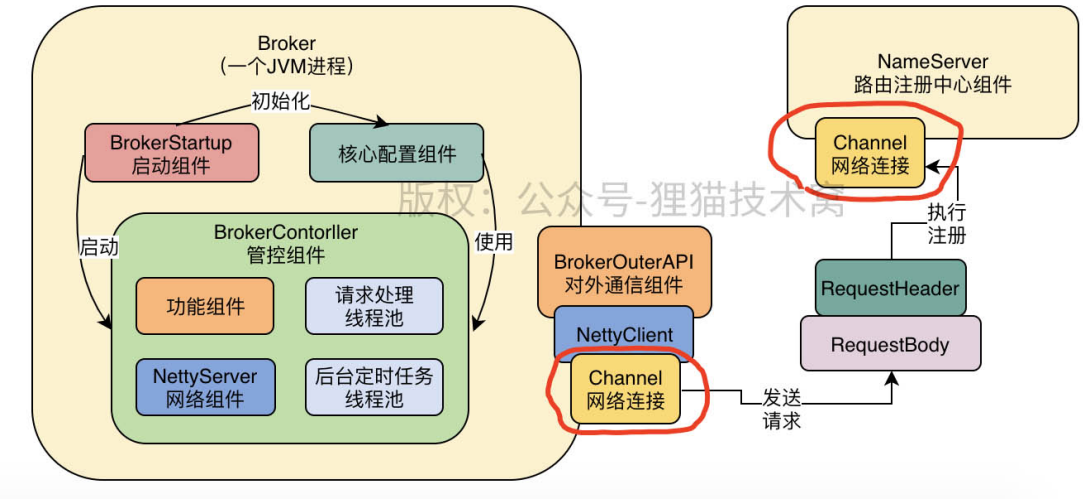
那我们接着看下面的this.createChannel(addr)方法是如何实际通过一个NameServer的地址创建出来一个网络连接的吧。

我们看下面的代码，我写了详细的注释，大家仔细看里面的注释。

```
private Channel createChannel(final String addr) throws InterruptedException {

    // 这个地方就是在尝试获取缓存里的连接，如果有缓存就返回连接了
    ChannelWrapper cw = this.channelTables.get(addr);

    if (cw != null && cw.isOK()) {
        return cw.getChannel();
    }

    if (this.lockChannelTables.tryLock(LOCK_TIMEOUT_MILLIS, TimeUnit.MILLISECONDS)) {
        try {
            // 下面一大坨代码也是在获取缓存里的连接
            boolean createNewConnection;
            cw = this.channelTables.get(addr);
            if (cw != null) {

                if (cw.isOK()) {
                    return cw.getChannel();
                } else if (!cw.getChannelFuture().isDone()) {
                    createNewConnection = false;
                } else {
                    this.channelTables.remove(addr);
                    createNewConnection = true;
                }
            } else {
                createNewConnection = true;
            }
            if (createNewConnection) {
               // 其实这里才是真正创建连接的地方
               // 大家可以看到，这里基于Netty的BootStrap这个类的connect()方法
               // 就构建出来了一个真正的Channel网络连接！
                ChannelFuture channelFuture = this.bootstrap.connect(RemotingHelper.string2SocketAddress(addr));
                log.info("createChannel: begin to connect remote host[{}] asynchronously", addr);
                cw = new ChannelWrapper(channelFuture);
                this.channelTables.put(addr, cw);
            }
        } catch (Exception e) {
            log.error("createChannel: create channel exception", e);

        } finally {
            this.lockChannelTables.unlock();
        }
    } else {
        log.warn("createChannel: try to lock channel table, but timeout, {}ms", LOCK_TIMEOUT_MILLIS);
    }

```

```
47    // 下面一大坨代码，无非就是把Channel连接返回回去了
48    if (cw != null) {
49
50        ChannelFuture channelFuture = cw.getChannelFuture();
51        if (channelFuture.awaitUninterruptibly(this.nettyClientConfig.getConnectTimeoutMillis())) {
52            if (cw.isOK()) {
53                log.info("createChannel: connect remote host[{}] success, {}", addr, channelFuture.toString());
54                return cw.getChannel();
55
56            } else {
57                log.warn("createChannel: connect remote host[" + addr + "] failed, " + channelFuture.toString(), channelFuture.cause());
58            }
59        } else {
60            log.warn("createChannel: connect remote host[{}] timeout {}ms, {}", addr, this.nettyClientConfig.getConnectTimeoutMillis(),
61                channelFuture.toString());
62        }
63    }
64    return null;
65 }
```

大家看下图，只要上面的Channel网络连接建立起来之后，我下面画红圈的地方，其实Broker和NameServer都会有一个Channel用来进行网络通信。



4、如何通过Channel网络连接发送请求?

接着我们看看，如何通过Channel网络连接发送请求出去?

其实核心入口就是下面的方法，之前讲过了。

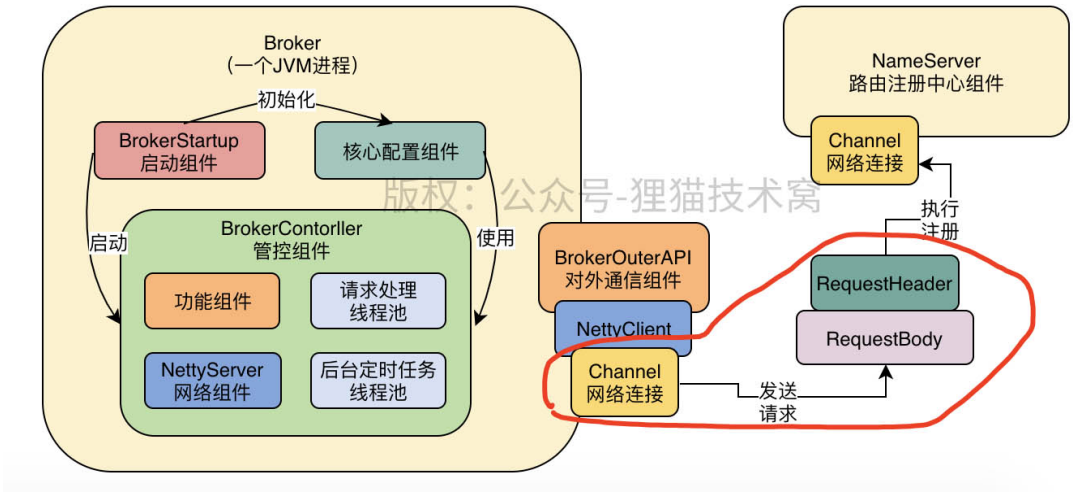
RemotingCommand response = this.invokeSyncImpl(channel, request, timeoutMillis - costTime);

我们进入这个方法去看看，他是如何发送网络请求出去的？我同样写了详细的注释，大家注意看我的注释就行，一些乱七八糟的代码如果暂时看不明白也没关系的，关键是抓住重点的逻辑。

```java
public RemotingCommand invokeSyncImpl(final Channel channel,
                                      final RemotingCommand request,
                                      final long timeoutMillis) {

    final int opaque = request.getOpaque();
    try {
        final ResponseFuture responseFuture =
            new ResponseFuture(channel, opaque, timeoutMillis, null, null);

        this.responseTable.put(opaque, responseFuture);
        final SocketAddress addr = channel.remoteAddress();

         // 前面的代码你不用过多的关注，重点是下面的代码
         // 你会发现，基于Netty来开发，核心就是基于Channel把你的请求写出去
        channel.writeAndFlush(request).addListener(new ChannelFutureListener() {

            @Override
            public void operationComplete(ChannelFuture f) throws Exception {

                if (f.isSuccess()) {
                    responseFuture.setSendRequestOK(true);
                    return;
                } else {
                    responseFuture.setSendRequestOK(false);
                }
                responseTable.remove(opaque);
                responseFuture.setCause(f.cause());
                responseFuture.putResponse(null);
                log.warn("send a request command to channel <" + addr + "> failed.");
            }
        });

        // 在下面这个地方，其实核心就是在等待请求的响应结果回来
        RemotingCommand responseCommand =
            responseFuture.waitResponse(timeoutMillis);

        if (null == responseCommand) {
            if (responseFuture.isSendRequestOK()) {
             ...
            } else {
              ...
            }
        }
        return responseCommand;
    } finally {
        this.responseTable.remove(opaque);
    }

}
```

其实上面的代码只要关注我写的几行注释就可以了，抓住重点，就知道，最终底层其实就是基于Netty的Channel API，把注册的请求给发送到了NameServer就可以了。

我们看下面的图，里面的红圈就展示了通过Channel发送网络请求出去的示意。



5、今日源码分析作业

希望大家参考今天的文章，把Broker注册的时候，在NettyClient底层进行Channel网络连接建立，以及通过Channel连接把注册请求发送出去的这些逻辑，都自己看一遍，同时好好理解我文章里画出来的图。

大家一定要注意，看源码的每一行细节是一个过程，加深你的理解，但是最终记在你脑子里的，一定是一幅一幅的图，这才是最终你自己沉淀下来的东西

如果大家有什么源码分析心得，可以发在评论区进行交流。

End

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情留言提问，我会逐一答疑

如何加群：购买狸猫技术窝专栏的小伙伴都可以加入狸猫技术交流群，一个非常纯粹的技术交流的地方

具体加群方式，请参见目录菜单下的文档：《付费用户如何加群》（**购买后可见**）